Redis+R = Multi-user happiness

Abstract

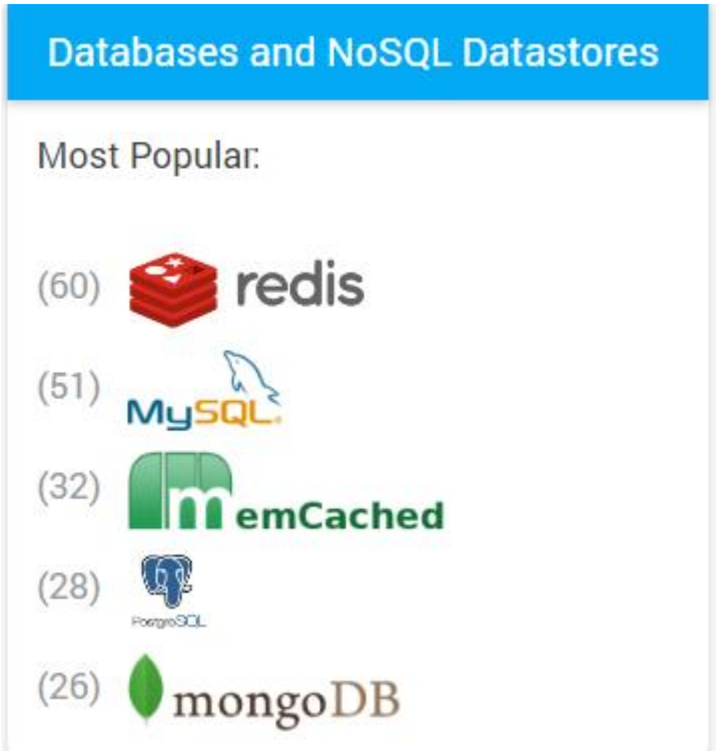
- There are many options for data persistence from R; from SQL server to Mongo but one option that is fast, powerful, rich and very well suited to R programming is Redis. The combination of data structures like queues, ordered lists, hash sets with a light in-memory footprint makes Redis the ideal choice for apps that have a high transaction rate, and many users. In this tutorial, we will show how easy is it is to build R applications with Redis and in particular, how Shiny apps can share back end data through a Redis interface.

Who's using Redis?

A list of well known companies using Redis:

Twitter GitHub Weibo Pinterest Snapchat Craigslist Digg StackOverflow Flickr

Stack crunch



Databases and NoSQL Datastores

Most Popular:

(60) redis

(51) MySQL

(32) MemCached

(28) PostgreSQL

(26) mongoDB

Why is Redis different from MSSQL and MySql

- No schema. Quick structure changes. Only once.
- Everything that is stored is a string that is serialized an deserialized by the calling system.
- Redis is fast (About 70 000 transactions a second on a standard server) and predictable (biggest problem with SQL is unexpected delays).
- Redis data can be easily split across servers (clusters)
- Redis is free.

Why is Redis different from Mongo etc

- Redis is not a *plain* key-value store, actually it is a *data structures server*, supporting different kind of values.
- What this means is that, while in traditional key-value stores you associated string keys to string values, in Redis the value is not limited to a simple string, but can also hold more complex data structures.
- This functionality allows building of complex queue multi user apps easily

Redis data types

1. **Binary safe strings:** can be used to store any single R values
2. **Lists:** collections of string elements sorted according to the order of insertion. They are basically *linked lists*. Also various specialisations for messaging and use as stacks.
3. **Sets:** collections of unique, unsorted string elements.
4. **Sorted sets:** similar to Sets but where every string element is associated to a floating number value, called *score*. The elements are always taken sorted by their score, so unlike Sets it is possible to retrieve a range of elements (for example you may ask: give me the top 10, or the bottom 10).
5. **Hashes:** which are maps composed of fields associated with values. Both the field and the value are strings. This is very similar to Ruby or Python hashes.
6. **Bit array:** array of bits. You can set and clear individual bits, count all the bits set to 1, find the first set or unset bit, and so forth.
7. **HyperLogLogs**: a probabilistic data structure which is used in order to estimate the cardinality of a set without storing all elements. Not implemented by rredis.

Redis resources

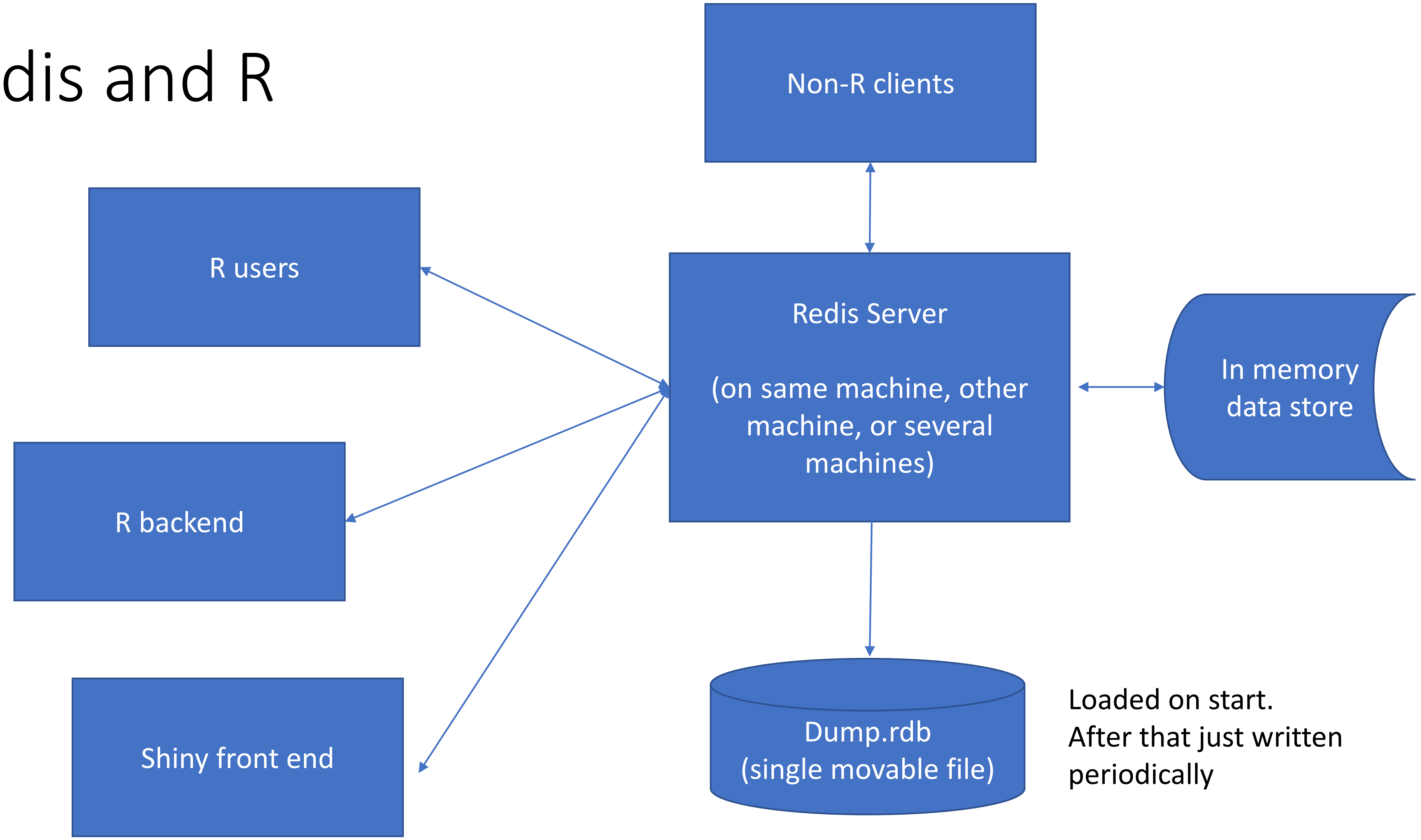
- Thanks to:
  - Salvatore Sanfilippo – Creator of Redis
  - Bryan W. Lewis blewis@illposed.net – Author of rredis package

- Download:
  - Linux - https://redis.io/download
  - Windows - https://github.com/MSOpenTech/redis/releases
- Quick tutorial: https://redis.io/topics/data-types-intro
- Longer tutorial: https://www.tutorialspoint.com/redis/
- Easy to read and short free book: http://openmymind.net/2012/1/23/The-Little-Redis-Book/
- Lots of others at: https://redis.io/documentation

The rredis package

- The rredis Package on cran

Redis and R


Non-R clients
R users
Redis Server
(on same machine, other machine, or several machines)
In memory data store
R backend
Shiny front end
Dump.rdb
(single movable file)
Loaded on start.
After that just written periodically

Can Redis perform better than R, .Rdata

Redis and Data.Tables

- Two ways to store data tables
  - A: Just as a big blob in single redis key
  - B: Use a Hash set with and id or row number as the key and each row as a single row data table as the value.
- A: easier but not good if you are sharing data or if the data table is large.
- B: nice if you need to modify a row at a time and want to allow multiple users to modify/use the table.